DETERMINISTIC ERROR RECOVERY PROTOCOL

BACKGROUND OF THE INVENTION

Field of the Invention

[0001] This invention generally relates to the field of error recovery, and more particularly, the invention relates to a procedure that is very well suited for error recovery across long communication lines. Even more specifically, the invention relates to an error recovery protocol that is particularly well adapted for use with massively parallel computers used for various applications such as, for example, applications in the field of life sciences.

Background Art

[0002] Figure 1 illustrates a pair of communication nodes A and B, each of which has a sender and a receiver. These two such nodes are connected with two cables. Each wire connects a sender/receiver pair. The cables may be long in the sense that a bit of data takes many clock cycles to traverse the cable. This type of hardware is encountered in many applications and most notably in massively parallel supercomputers.

[0003] Obviously, as with any communication channel, errors can occur during communication. Assuming that the receivers have the capability of detecting such errors, a protocol is needed in order to ensure that both nodes recover from the error correctly and resume communication without any data loss. If there are no extra sideband cables to communicate recovery signals, this is a difficult task since the original cables must be used. In doing so, one is exposed to errors in the recovery signals themselves. Although error recovery methods that solve this problem exist, they have the disadvantage that they do not put the system of two nodes into a known state and that they depend on time-out and specific data sequence methods.

SUMMARY OF THE INVENTION

[0004] An object of this invention is to provide a procedure for recovering when one or both of a pair of connected communication nodes encounters an error.

[0005] Another object of the present invention is to provide an error recovery procedure that is effective across long communication lines.

[0006] A further object of the invention is to put both of a pair of nodes that communicate with each other into a known state after one or both nodes encounter an error.

[0007] Another object of this invention is to put both of a pair of nodes into a known state after one or both of the nodes encounter an error, and to do this independently of how many further errors are encountered during recovery (provided the number of errors is not infinite) and without requiring any special data sequences or time outs.

[0008] These and other objectives are obtained with an error recovery method and system for use with a communication system having first and second nodes, each of said nodes having a receiver and a sender, the sender of the first node being connected to the receiver of the second node by a first cable, and the sender of the second node being connected to the receiver of the first node by a second cable. The method comprises the step of, after one of the nodes detects an error, both of the nodes entering the same defined state. In particular, the receiver of the first node enters an error state, stays in the error state for a defined period of time T, and, after said defined period of time T, enters a wait state. Also, the sender of the first node sends to the receiver of the second node an error message for a defined period of time Te, and after the defined period of time Te, the sender of the first node enters an idle state.

[0009] The preferred embodiment of this invention, described in detail below, provides a protocol that is guaranteed to put both nodes into a known state after one or both nodes encounter an error. This is achieved independently of how many further errors are encountered during recovery (provided they are not infinite). Also, no special data sequences or time outs are used. These properties make this method robust.

[0010] Further benefits and advantages of the invention will become apparent from a consideration of the following detailed description, given with reference to the accompanying drawings, which specify and show preferred embodiments of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] Figure 1 illustrates a pair of connected communication nodes.

[0012] Figure 2 is a flow chart showing a preferred error recovery procedure of this invention.

[0013] Figure 3 is a time diagram showing various time periods used in the preferred embodiment of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0014] Figure 1 shows two nodes A, B connected with a pair of cables. Receiver A and B are identical. Also sender A and B are identical. For simplicity, assume that the cables transfer one byte of data at each clock cycle. The communication protocol is packet based with byte-long packet headers that must be recognized by the receivers and trailers that contain some type of packet integrity check (such as checksum or CRC). When the receivers receive a special type of byte, called IDLE, they recognize it and do nothing further. If a header byte is not of known type, or if the packet integrity check fails, the receiver goes into an ERROR state. During normal operation, if there are no packets to be sent, the senders transmit IDLE and the receivers are in their normal WAIT state where they listen to incoming traffic.

[0015] If either node gets an error (unknown header type or bad packet integrity check), the protocol of this invention ensures that at some later time both nodes will have their receivers in normal WAIT state and their senders sending IDLE. This will happen independently of how many errors are encountered until that state is reached (for as long as the number of errors is finite). After this, sender A and sender B enter their standard resend mode (not necessarily at the same time). In this protocol there is no timer in the sender that times out if a packet is not acknowledged within a certain time. A special byte is used, called ERROR_BYTE. This byte can be anything for as long as it is *not* a recognizable header type by the receiver.

[0016] The protocol implementation, with reference to Figure 2, is as follows:

[0017] Receiver:

[0018] 1) Wait for known header type. If the receiver gets anything else, it goes to ERROR_STATE.

[0019] 2) Normal state machine processing:
If the receiver gets any bad packet integrity check, it goes to ERROR_STATE.

[0020] 3) ERROR_STATE:
Do not listen to ANY incoming data.
Set errror_flag bit to 1.
Stay in error for T cycles and then go back to WAIT_STATE.

[0021] Sender:

[0022] 1) INITIAL_STATE:
If (error_flag = 1), then
  reset it to 0 and go to SEND_ERROR state
else
  operate normally.

[0023] 2) SEND_ERROR state:
Send the ERROR_BYTE for Te cycles (this will put the neighbor's receiver to ERROR_STATE)
If (error_flag = 1), then
  go to INITIAL_STATE state
else
  go to SEND_IDLE state.

[0024] 3) SEND_IDLE state:

Send IDLE for Ti cycles
If (error_flag = 1), then
go to INITIAL_STATE state
else
go to RESEND_STATE.

[0025] 4) RESEND_STATE:
Resend any data (including acknowledgements etc...) that may have been lost during recovery. Any suitable mechanism that keeps track of what has been successfully received by the corresponding receiver may be used. Such mechanisms are known in the art.
Go back to INITIAL_STATE.

[0026] In the above discussed scheme, there are 3 parameters:

[0027] T - the number of cycles the receiver must stay in ERROR_STATE,

[0028] Te - the number of cycles the sender sends ERROR_BYTEs, and

[0029] Ti - the number of cycles the sender sends IDLE_BYTEs.

[0030] These parameters do not need to be hard-wired but they can be set in registers by the software. This will give additional flexibility.

[0031] The Te can be small. The purpose of the ERROR_BYTE is to put the neighbor's receiver into ERROR_STATE. Even if an ERROR_BYTE is corrupted, the neighbor's receiver will still go to ERROR_STATE. It may be noted that counting is not required here. A single ERROR_BYTE will put the receiver into ERROR_STATE. If the first byte is mistaken for an IDLE, then the second ERROR_BYTE will accomplish the desired result and so on. Even if it is mistaken for a DATA_TYPE, the packet integrity will be wrong and will put the receiver into ERROR_STATE. The only fail mode of this protocol is a conspiracy that turns the Te ERROR_BYTEs into a non-error pattern. However, the choice of bits in the ERROR_BYTE is arbitrary for as long as the byte is not recognizable by

the receiver. Therefore, one can choose these bits to minimize the chance that a malfunction in the signaling technique transforms them to a known type. Furthermore, one can pick several error bytes, i.e. ERROR_BYTE_0, ERROR_BYTE_1, etc., and send them repeatedly. Also, since there is no restriction on Te, one can make this sequence as long as desirable. The important consideration is that any sequence of Te bytes that are not of known type can put the receiver into ERROR_STATE and this is all that is needed.

[0032] The Ti must be set at some large value. Specifically, this value needs to large enough to allow all previous packets and ERROR_BYTEs to get out of the cables and then some more (see calculation below).

[0033] When the receiver is in ERROR_STATE, it does not listen to anything. Therefore even if there are errors, they are fully ignored. When the receiver comes out after T cycles, it only gets IDLEs for a while and then the resend data. If, after it comes out, the receiver gets a bad IDLE or an error during resend, then the sequence is repeated (discussed further below).

[0034] An important feature is that there is a period during which both receivers are in WAIT state receiving IDLES and both senders are in IDLE_SEND state. This is a known state and it occurs before resend.

[0035] *Detailed calculation of the three parameters*:

[0036] T and Ti are calculated below. These calculations show that one can pick values for which the receiver goes back to WAIT_STATE after the stream of idles has started and before it ends (resend starts after the stream of IDLEs ends). The calculations also show that this is true independently of the state of the receiver/sender in either node. Also, Te can be set to any non-zero positive value.

[0037] Assume that node A goes into the error state before or at the same time as B. As will be apparent to those of ordinary skill in the art, the case with B going into error before A is the same but with A and B interchanged.

[0038] The times below are absolute and time zero is when A goes into ERROR_STATE. All counters increment once each cycle.

[0039] *For node A:*

[0040] T_A = 0

[0041] Receiver A goes to ERROR_STATE and sets eflag = 1. The receiver A counter starts counting from 0 up to T.

[0042] T_A = Ts_A

[0043] Sender finishes current task. This can happen immediately, in which case Ts_A=0, or after a maximum time that depends on the hardware design. In any case, this max value is deterministic and it does not change. That time interval is denoted by Tr.

[0044] Therefore 0 < Ts_A < Tr.

[0045] Sender A sees eflag=1 and goes to SEND_ERROR state. It resets eflag=0. Starts sending ERROR_BYTE bytes for Te cycles.

[0046] T_A = Ts_A + Te

[0047] Sender A goes to SEND_IDLE state and starts sending IDLE for Ti cycles.

[0048] T_A = Ts_A + Te + Ti

[0049] Sender A stops sending IDLE and goes to the resend state and starts the resend sequence.

[0050] *For node B:*

[0051] $T\_B = T0\_B$

[0052] Receiver B goes to ERROR_STATE and sets eflag <= 1. The receiver B counter starts counting from 0 up to T. T0_B can be as small as zero if node B got an error at exactly the same time as node A. T0_B can be as large as the time since the last ERROR_BYTE was received by node B.

[0053] $0 < T0\_B < Ts\_A + Te + Tc$

[0054] where Tc is a time larger than the time it takes one byte to traverse the longest cable in the network.

[0055] $T\_B = T0\_B + Ts\_B$

[0056] Sender finishes current task. As described above, $0 < Ts\_B < Tr$.

[0057] Sender B sees eflag=1 and goes to SEND_ERROR state. Sender B resets eflag=0. Starts sending ERROR_BYTE bytes for Te cycles.

[0058] $T\_B = T0\_B + Ts\_B + Te$

[0059] Sender B goes to SEND_IDLE state and starts sending IDLE for Ti cycles.

[0060] $T\_B = T0\_B + Ts\_B + Te + Ti$

[0061] Sender A stops sending IDLE and goes to the resend state and starts the resend sequence.

[0062] *To summarize*:

[0063] $0 < Ts\_A < Tr$

[0064] $0 < Ts\_B < Tr$

[0065] Tr = max time for sender to come back to its initial state.

[0066] Tc = is a time larger than the time it takes one byte to traverse the longest cable in the network.

[0067] $0 < T0\_B < Ts\_A + Te + Tc$

[0068] T0_B is the time the counter of receiver B starts.

[0069] From the above, one has that for receiver A:

[0070] First IDLE arrives at:

[0071] $T\_A = T0\_B + Ts\_B + Te + Tc$

[0072] And last IDLE arrives at:

[0073] $T\_A = T0\_B + Ts\_B + Te + Tc + Ti$

[0074] Also from the above, one has that for receiver B:

[0075] First IDLE arrives at:

[0076] $T\_B = Ts\_A + Te + Tc$

[0077] And last IDLE arrives at:

[0078] $T\_B = Ts\_A + Te + Tc + Ti$

[0079] Therefore the constraint for the parameter T for node A is:

[0080] $T0\_B + Ts\_B + Te + Tc < T < T0\_B + Ts\_B + Te + Tc + Ti$

[0081] And the constraint for node B is:

[0082] $Ts\_A + Te + Tc < T + T0\_B < Ts\_A + Te + Tc + Ti \Rightarrow$

[0083] $Ts\_A + Te + Tc - T0\_B < T < Ts\_A + Te + Tc + Ti - T0\_B$

[0084] These constraints must be satisfied at the same time. Also, they must be satisfied for any value of Ts_A, Ts_B in the ranges given above. Therefore one must replace the left hand sides with the max value they can have and the right hand sides with the min value they can have. Then from the two inequalities, one must pick the max left hand side and the min right hand side. One gets:

[0085] $2(Te + Tc + Tr) < T < Ti - Tr$ (Equation 1).

[0086] Therefore one should pick T as in equation (1) above and Ti must satisfy:

[0087] $2\,Te + 2\,Tc + 3\,Tr < Ti$ (Equation 2).

[0088] Obviously equation 1 has a solution for T.

[0089] Now, there is a further constraint in Ti. Since receiver A will start listening at time $T\_A = T$ but receiver B will not start listening until time $T\_B = T0\_B + T$, there is an interval T0_B that A can receive a "bad" IDLE and go back into ERROR state before the "known state" described above is entered (both receivers in normal wait and both senders in normal send). Then the sender A may resend the ERROR_BYTEs during a time that B is not listening. If this happens, B will continue and receiver B will come out of ERROR_STATE. Also, sender B will continue and will start the resend sequence at time $T\_B = T0\_B + Ts\_B + Te + Ti$. This will arrive at A a little later, but consider the worst case of zero length wire.

So, this will arrive at A at time T0_B + Ts_B + Te + Ti. Receiver A needs to be back at listening at this time.

[0090] Receiver A will be back at listening at time $2T < T\_A < 2T + T0\_B$. Therefore we need:

[0091] $2T + T0\_B < T0\_B + Ts\_B + Te + Ti \Rightarrow$

[0092] $2T - Ts\_B - Te < Ti$ For example, this is satisfied if:

[0093] $2T < Ti$ (Equation 3).

[0094] Equation 3 is consistent with equation 1 and therefore there are settings for which the known state (both receivers in normal wait and both senders sending IDLEs) is reached.

[0095] *Parameter values*:

[0096] An example of settings is given below:

[0097] Assume that

[0098] $2\,Tc = Tr$

[0099] Also, pick Te to be any number reasonably large. For simplicity let also

[00100] $2\,Te = Tr$

[00101] Of course this is unnecessarily large since Te needs only be more than enough to put the receiver into error state.

[00102] If one picks:

[00103] T = 5 Tr

[00104] Ti = 12 Tr

[00105] then equations 1 and 3 are satisfied.

[00106] In terms of Tc:

[00107] Te = Tc

[00108] T = 10 Tc

[00109] Ti = 24 Tc

[00110] *Time diagram:*

[00111] The time diagram for the above example is given in Figure 3.

[00112] Time is from top to bottom. The * indicate time lines that can be shrunk to 0 but cannot be larger than the *** indicate. The | indicate time lines that are fixed. The numbers on the left indicate units of Tr.

[00113] While it is apparent that the invention herein disclosed is well calculated to fulfill the objects stated above, it will be appreciated that numerous modifications and embodiments may be devised by those skilled in the art, and it is intended that the appended claims cover all such modifications and embodiments as fall within the true spirit and scope of the present invention.